Assembler Documentation

This application helps convert Assembly language into hex code for our microprocessor to understand.

There are two main categories of instructions, namely;

* ALU operations
* Miscellaneous operations

Syntax for ALU operations

Under the ALU operations we have 8 different operations, namely”

{ADD, SUB, MULT, DIV, SHL, SHR, AND, OR}.

The general syntax for an ALU operation is,

ALU RA,RB,RC

Where:

“**ALU** “ : This field is for the opcode

“**RA** “ : This field is for the first register

“**RB** “ : This field is for the second register

“**RC** “ : This field is for the destination register

Below are examples of how each of the ALU operations are implemented using the syntax stated above.

Example [ ADD ] :

The operation I want to perform is:

2 + 3 = 5

So, in assembly representation,

ADD R1,R2,R3

Where:

**ADD** is the operation being performed on the numbers

**R1** contains the number 2

**R2** contains the number 3

**R3** is the register that will contains the result of the operation

Example [ SUB ] :

The operation I want to perform is:

6 - 2 = 5

So, in assembly representation,

SUB R1,R2,R3

Where:

**SUB** is the operation being performed on the numbers

**R1** contains the number 6

**R2** contains the number 2

**R3** is the register that will contains the result of the operation

Example [ MULT ] :

The operation I want to perform is:

2 x 3 = 6

So, in assembly representation,

MULT R1,R2,R3

Where:

**MULT** is the operation being performed on the numbers

**R1** contains the number 2

**R2** contains the number 3

**R3** is the register that will contains the result of the operation

Example [ DIV ] :

The operation I want to perform is:

4 ÷ 2 = 2

So, in assembly representation,

DIV R1,R2,R3

Where:

**DIV** is the operation being performed on the numbers

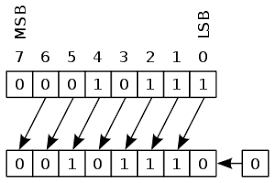
**R1** contains the number 4

**R2** contains the number 2

**R3** is the register that will contains the result of the operation

Example [ SHL ] :

The operation I want to perform is:



Using a logical shift left so, in assembly representation,

SHL R1,R2,R3

Where:

**SHL** is the operation being performed on the numbers

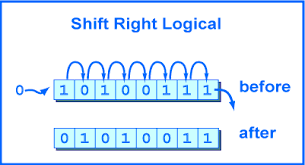
**R1** contains the number 4 which is the value we want to do the operation on

**R2** contains a number that tell the processor how many times to shift left. In this case the value will contain 1 in order to perform our required operation

**R3** is the register that will contains the result of the operation

Example [ SHR ] :

The operation I want to perform is:



Using a logical shift right so, in assembly representation,

SHR R1,R2,R3

Where:

**SHR** is the operation being performed on the numbers

**R1** contains the number 4 which is the value we want to do the operation on

**R2** contains a number that tell the processor how many times to shift right. In this case the value will contain 1 in order to perform our required operation

**R3** is the register that will contains the result of the operation

Example [ AND ] :

The operation I want to perform is:

2 & 3 = 2

So, in assembly representation,

AND R1,R2,R3

Where:

**AND** is the operation being performed on the numbers

**R1** contains the number 2

**R2** contains the number 3

**R3** is the register that will contains the result of the operation

Example [ OR ] :

The operation I want to perform is:

2 OR 3 = 3

So, in assembly representation,

ADD R1,R2,R3

Where:

**OR** is the operation being performed on the numbers

**R1** contains the number 2

**R2** contains the number 3

**R3** is the register that will contains the result of the operation

Syntax for MISC. operations

Under the MISC operations we have 8 different operations, namely”

{MOVR, MOVI, LOAD, STORE, JMP, JMPZ, JMPN, HALT/NOOP }.

* **MOVR**

This operation allows us to move the contents from one register to another register

The syntax for a MOVR operation is,

MOVR RA,RB

Where:

“**MOVR** “ : This field is for the opcode

“**RA** “ : This field is for the SOURCE register

“**RB** “ : This field is for the DESTINATION register

* **MOVI**

This operation allows us to move an immediate value into a register

The syntax for a MOVI operation is,

MOVI RA,IMME

Where:

“**ALU** “ : This field is for the opcode

“**RA** “ : This field is for the DESTINATION register

“**IMME** “ : This field is for the immediate value

* **LOAD**

This operation allows us to load contents in memory into a register

The syntax for a MOVI operation is,

LOAD RA,ADDR

Where:

“**ALU** “ : This field is for the opcode

“**RA** “ : This field is for the DESTINATION register

“**ADDR** “ : This field is for the address location of the data in memory.

* **STORE**

This operation allows us to store the data from a register into memory.

The syntax for a STORE operation is,

STORE RA,ADDR

Where:

“**STORE** “ : This field is for the opcode

“**RA** “ : This field is for the SOURCE register

“**ADDR** “ : This field is for the address location to store the value

* **JMP**

This operation allows us to jump to a specified instruction location in memory.

The syntax for a JMP operation is,

JMP ADDR

Where:

“**JMP** “ : This field is for the opcode

“**ADDR** “ : This field is for the address location of the instruction to jump to

* **JMPZ**

This operation allows us to jump to a specified instruction location in memory if the zero flag is enabled,

The syntax for a JMPZ operation is,

JMPZ ADDR

Where:

“**JMPZ** “ : This field is for the opcode

“**ADDR** “ : This field is for the address location of the instruction to jump to

* **JMPN**

This operation allows us to jump to a specified instruction location in memory if the negative flag is enabled.

The syntax for a JMPN operation is,

JMPN ADDR

Where:

“**JMPN** “ : This field is for the opcode

“**ADDR** “ : This field is for the address location of the instruction to jump to

* **NOP**

This opcode means no operation. That is no operation is to be done.

The syntax for a NOP operation is,

NOP

Where:

“**NOP** “ : This field is for the opcode

* **HALT**

This opcode halts the microprocessor

The syntax for a HALT operation is,

HALT

Where:

“**HALT** “ : This field is for the opcode